Gabriel Lorvan

Software Development I

Final Project Writeup

Abstract:

This project I took on is a method to send files between a client and a remote server and visa versa. I did have to poke around online to see what methods and classes were out there with which I was unfamiliar, but I learned quite a bit from that experience!

Introduction:

The idea for this project came from a free server I have up and running from Software Development II, which I am taking concurrently with this course. The system itself is simple: there is one batch of code that the sender runs and a similar batch of code that the receiver runs. I will go into more detail in a moment. The problem I am attempting to solve is a workaround for Git, which is what I am currently using to update files on the server. Of course, this project is not the first of its kind; there are many more systems out there that do the same thing. I, however, do not know how all of them work, but I can describe how my system works, which I will do later.

Detailed System Description:

As mentioned earlier, this system works between two or more users. One, the sender, runs the batch of code that prepares to send the desired file by essentially broadcasting on a port of his own computer. The receiver then runs a similar batch of code that connects to the sender's computer via that port and downloads the file being broadcast. The only potential drawbacks to this system are security, as the broadcast file is readily available to anyone if they know where to look, and speed, as sending multiple files requires multiple connections.

UML diagrams:

Sender
+SOCKET_PORT: int
+FILE_TO_SEND: String
+main(String[] args): void

Receiver
+SOCKET_PORT: int
+SERVER: String
+FILE_TO_RECEIVE: String
+FILE_SIZE: int
+main(String[] args): void

Although the UML diagrams are simple, there is a lot going on in each main method.

Requirements:

As mentioned above, the main reason I chose this project was to make the process of transferring files to the remote server a bit more intuitive and less convoluted than uploading the desired files to GitHub and downloading them onto the server or SCPing (secure copying) them from my hard drive to the server. Although both of these options work (as well as several others, which I will discuss later), I wanted to make an easier method of which I more thoroughly understood the inner workings. So, in essence, there is not really a problem, as I can already send files to the remote server, this project serves to further my understanding of how systems such as this work.

Literature Survey:

As just discussed, there are methods of transporting files from one host to another already available. One of the better-known methods is FTP (File Transfer Protocol), one of the most basic and ingrained systems available in almost all computers. FTP is exactly what its name suggests, allowing one host to send files to another host. FTP, however, is not terrifically secure, requires intimate understandings of its command prompt, and behaves differently with almost every client and server. There are a host (no pun intended) of other issues with FTP, from packets being sent out of order to firewalls connecting it to the wrong port because of misread encryptions.

Separate from FTP, SFTP (Secure File Transfer Protocol) offers an extra layer of security from its underlying protocol SSH (Secure Shell) and many other capabilities including resumption of interrupted transfers, directory listings, and remote file removal. SFTP is also designed to be platform-independent. That said, it still requires knowledge of its command line interface.

SCP (secure copy) is similar to SFTP but is usually only used on Unix devices (like my remote server). It offers some capabilities that SFTP does not, like remote-to-remote mode, but offers very similar functionality.

GitHub, as we all know, is GitHub, which allows for uploading and downloading files to hosts as well as storage in the cloud for future use and general access if desired.

User Manual:

The code, although it looks blocky and complicated, is surprisingly simple from the users' perspective. The users on either end of the code need simply to provide a few pieces of information to the code, and the code is automated to do the rest on its own:

The sender's code prompts the user to enter the path name of the file he wants to send, sets up a socket using his computer's IP address and a preselected port number, and waits for the receiver to connect. The receiver's code (when run in conjunction with the sender's code) prompts the user to enter the IP address of the sender (for some minor security) and the name of the expected file (to ensure that the file type and remain correct). The receiver's code then connects to the socket on the sender's computer. The sender's code then processes the file into bits and passes those bits to the receiver's code, which puts the bits back together. The sender's code then flushes the data stream and closes itself, as the file has been received. The receiver's code, when it finished parsing the data, does the same.

Of course, Java is required on both ends or else the code cannot run. Furthermore, the sender's code should be run before the receiver's code or the receiver's code will time out and throw an error because it had nothing to connect to. The sender's code is not automated to time out, so it will continue to broadcast the file on the socket until the code is terminated or the receiver connects and receives the file. The file size limit is just over 6 megabytes arbitrarily, but that can be changed by changing the FILE_SIZE constant in the receiver's code if need be.

Conclusion: Summarizes the goals accomplished by the system.

This system accomplished all of the goals I set out to accomplish. It allowed me a closer look at how file transfer protocols of all sorts run. Not only that, but it offers a simpler method of sending and

receiving files that avoids the necessity of learning yet another command line language and uploading and downloading via GitHub.

References/Bibliography:

FTP issues: http://etutorials.org/Networking/Check+Point+FireWall/Chapter+6.+Common+Issues/Problems+with+FTP/

Source code off of which my code is based: http://www.rgagnon.com/javadetails/java-0542.html

SFTP: https://en.wikipedia.org/wiki/SSH_File_Transfer_Protocol

SCP: https://en.wikipedia.org/wiki/Secure_copy